Description

BACKGROUND OF THE INVENTION:

1. Field of the Invention.

The invention relates to the field of filtering of video signals for a raster scanned display, particularly one employing computer generated pixel data.

2. Prior Art.

Most cathode ray tube (CRT) computer video displays are formed with a raster scan. Many of the standards used with these displays can be traced to television standards. For example, two interlaced fields are frequently used to form a frame. Many early personal computers provided compatible NTSC signals to permit a user to use low cost television receivers. In other instances, computers generate signals such as overlays which are used in conjunction with NTSC signals. Thus, personal computers often generate pixel data for use on interlaced, raster-scanned displays.

Computer generated data has some characteristics which make it less desirable for an interlaced, raster-scanned display than video signals originating in a video camera. For example, pixel data can exhibit changes (e.g., amplitude) over its entire range from pixel-to-pixel. That is, virtually any change in pixel data can occur from one pixel to the next. In contrast, video data from a traditional video camera uses a beam spot which encompasses more than a single pixel area. The data interpreted for a single pixel in this case takes into account to some extent the intensity and color of the surrounding area. Therefore, there is a softening, even a blurring, that occurs as the beam scans the image in a camera.

The human visual system is an edge-detection system. The eyes are very good at finding contours that delineate shapes. To give an example, when displaying a sequence of adjacent gray bars of increasing density on a computer display, the edges between the bars seem emphasized. Perceptually the gray bars do not look like solid colors, but rather they look like they have been shaded between their edges. In other words, the border between the gray bars appear enhanced by the edge-detection mechanisms of the eye.

When a typical real world scene is displayed on an interlaced display, there are no abrupt transitions from one scan line to the next. Objects generally do not have very hard edges, and those that do usually do not have edges lined up with a scan line. The result is the eye cannot find an edge from one scan line to the next.If the eye cannot find an edge between one scan line and the next, it cannot distinguish between lines. In an interlaced display a complete frame is drawn each 1/30th of a second, however, because of the interlacing each 1/60th of a second, either a given scan line or the next scan line is flashed. The eye perceives these multiple scan lines as thick single lines flashing at a 60 frame/second rate even though they are in fact flashing at 30 frames/second. By this model, close viewing of an interlaced display should result in perception of flicker at 30 frames/second.

This is in fact what happens; if one is close enough to view individual scan lines on a NTSC television, interlace flicker (i.e., 30 frame/second flashing) is seen, even with a real world image.

In the case of a computer generated image such as a MACINTOSH computer image on a interlace display, virtually every place where there is other than solid white or solid black there are abrupt transitions in the vertical dimension.

(Macintosh is a registered trademark of Apple Computer, Inc.) In the case of the "racing stripes" (alternately black and white horizontal lines) on the top of a typical Macintosh window, there is the most abrupt transition possible, black to white, stretched across the length of the window and repeated for several lines. Here, it is easy for the human eye to detect the edge from one scan line to the next, so it considers the scan lines as individuals, flashing at 30 frames/second. The visual perception of the human observer is that where there are abrupt transitions on the display, the NTSC image flickers noticeably enough to be distracting.

One additional subtlety is worth mentioning. The human eye will see flicker display wherever there are transitions (i.e.,edges) in the vertical dimension. But, the degree of flicker is not uniform for each type of graphic pattern. The worst pattern is the racing stripes across the top of a window, mentioned above. Text and other random patterns flicker as well, but not nearly as severely. This is accounted for by the fact that it is easier to discern vertical edges where there is a high horizontal correlation to the pattern (as in the case of the racing stripes), but harder to find the edges when there is a low horizontal correlation (as in the case of text).
As will be seen, since the present invention provides adaptive filtering for the subtlety.)

Numerous prior art techniques are known including those employing anti-aliasing filters for removing this flicker. In some cases, fiiters duplicate the softening effects of the camera beam, that is, pixel data for a cluster or spot of pixels is "averaged" or "convolved" to produce filtered pixel data. In general, these techniques require considerable computational overhead.

US-A-4 215 414 teaches the use of both vertical and horizontal filtering of pixel data to provide the blurring, a desirable characteristic inherent in a TV camera, and in particular a two-line convolution for phosphorous tubes. Alternate embodiments are also described, including the filtering of a 3x3 array of pixels or of a 3x2 array.

However, this reference recommends to store an entire line in a delay line, which is costly and therefore disadvantageous.

Further, to the extent that it does discuss vertical

filtering without delay lines, the reference provides the teaching of multiplexed reading of pixel data and then demultiplexing for presentation to an adder.

However, there is no suggestion in the description of this reference on how to obtain a practical system for vertical filtering without delay lines. For instance, it is far from obvious to implement the multiplexing and the demultiplexing with ordinary DRAMs, typically used in a frame buffer. This would require reading data from the memory at a rate too fast to enable providing a real time quality display. For example, if one assumes a dot clock rate of 12MHz and 3 accesses per pixel (for a pixel in three different lines), this requires an access time of approximately 28 nanoseconds. This access time is not achievable with ordinary DRAMs. If one were to select VRAMs with their greater bandwith, then the multiplexing and demultiplexing would not work since one could not arbitrarily pick the needed infonnation from the bit streams.

Therefore, US-A-4 215 414 simply does not provide any teaching for vertical filtering which can be practically implemented.

The present invention seeks to provide a method for providing filtered pixel data (in the vertical direction only) by means of a convolution technique which is capable of being performed "on the fly", by having computational demands which are substantially less than that required by the prior art systems.

SUMMARY OF THE INVENTION

The present invention provides a method as defined in claim 1. Preferred aspects of the invention are defined in the dependent claims.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a general block diagram illustrating the general placement of the present invention in a video system.

Figure 2 is a diagram used to illustrate a method used by the present invention to read data from a frame buffer.

Figure 3 is a diagram used to illustrate an alternate method used by the present invention to read data from a frame buffer.

Figure 4 is a block diagram illustrating an embodiment of a convolver used in the present invention

Figure 5 is a block diagram illustrating another embodiment of a convolver used in the present invention.

Figure 6 is a block diagram illustrating another method for obtaining convolved data particularly useful where not many bits are stored for each pixel.

Figure 7A is a block diagram of a general prescaler which can be used with the convolver of the present invention.

Figure 7B is a block diagram of another prescaler which can be used with the convolver of the present invention.

Figure 8 illustrates a first step in implementing the present invention in software for a "chunky" frame buffer.

Figure 9 illustrates a second step in the implementation described in conjunction with Figure 8.

Figure 10 illustrates a third step in the implementation described in conjunction with Figures 8 and 9.

Figure 11 illustrates a fourth step in the implementation described in conjunction with Figures 8-10.

Figure 12 illustrates a fifth step in the implementation described in conjunction with Figures 8-11.

Figure 13 illustrates gray values loaded into the color lookup table.

DETAILED DESCRIPTION OF THE PRESENT INVENTION

A method for providing filtering data, in a raster-scanned video apparatus is described. The invention provides filtering in the vertical direction (perpendicular to the direction of the scan lines). In the following description numerous specific details are set forth in order to provide a better understanding of the present invention. It will be obvious, however, to one skilled in the art that the present invention may be practiced without these details. In other instances, well-known circuits and computer operations have been shown in block diagram form, in order not to obscure the present invention in unnecessary detail.

OVERVIEW OF THE PRESENT INVENTION

Referring first to Figure 1 a frame buffer 10 is illustrated which in the presently preferred embodiment may be an ordinary frame buffer, for example, one fabricated from dynamic random-access memories (DRAMS) or video random access memories (VRAMs). Most often, data is organized in the frame buffer by scan lines with data being stored for each pixel along each scan line. In some cases, the pixel data is organized in planes such that the pixel data for a given pixel is stored with each bit on a different one of the planes. When data is organized in this arrangement, a scanning address causes a bit from each plane to be read from the memory, the bits are assembled to form a pixel, and hence, the data for a given pixel is read from the memory for a video display. (Often when writing data to a memory organized by planes, an 8 or 16 bit word is written to each plane; this is particularly useful for a black and white, or two-color display, where only a single bit is stored per pixel and hence data is written into only a single plane). For some embodiments of the present invention, the data is stored in an ordinary manner as described above.

An address generator 11 is used to address the data in the buffer to provide an output signal for a video display. As will be seen with the present invention the order in which the data is scanned from the buffer is different

than that used in the prior art and hence, the address generator 11 provides this unique addressing order. (This is referred to as "kernel-scanned" in Figure 1.) The specific order employed will become apparent, particularly from the discussion below for Figures 2 and 3. Ordinary circuits may be used to implement the generator 11 and provide the order described in Figures 2 and 3. As in the case with prior art generators, the address generator 11 is generally synchronized with a dot clock.

The output from the buffer 10 is convolved by convolver 12. The output of the convolver 12 is pixel data which can be used in an ordinary manner for a video display. The convolver 12 is described in conjunction with Figures 7A and 7B.

In the currently preferred embodiment the output of the convolver 12 is gamma corrected. Such gamma correction is well-known in the art and used to compensate for the non-linear light intensity curve of CRT displays. The digital information on line 14 is converted to an analog form for coupling to a display.

In the following description it is assumed that the buffer 10 stores pixel data. It will be appreciated that the buffer may store pointers to another memory such as a color lookup table. In this event, the output of the buffer 10 is coupled to a color lookup table and the output of the color lookup table is coupled to the convolver 12.

In Figure 2 it is assumed that each of the blocks in the illustrated grid represents a pixel in a bit mapped buffer. In the horizontal direction the pixels are numbered from 0 through 9; it will be appreciated that in a typical memory, many more pixels are used in the display. In the vertical direction the rows of pixel data are numbered by scan line beginning with line 0, and ending at line 5. Again, it will be appreciated that in a typical display many more scan lines are used. Figure 2 thus represents the data organization that is found in a typical frame buffer.

For the present invention the data for a given pixel (e.g., pixel 0) is read from the memory for several lines (e.g., lines 1, 2 and 3) before the pixel data for pixel 1 is read from the memory. The pixel data for the several lines of a given pixel is convolved to provide the pixel data used by the display.

More specifically, in Figure 2 the data at locations 16, 17 and 18 is read from the memory before the data for pixel 19 is read from the memory. The pixel data from locations 16, 17 and 18 is then convolved to provide pixel data for pixel 0 of a display line. Next, the pixel data at locations 19, 20 and 21 is read from the memory and convolved to provide pixel data for pixel 1 of the display line. This process continues for each of the pixels 0 through 9 for scan lines 1-3 to provide pixel data for a given display line.

For the illustrated embodiment three lines of data are used in the convolution process. Any number of lines may in fact be used, for example, the data from lines n, n+1, n+2 ... N+n may be first used to provide pixel data for a first display line. Following this, the data from lines n+1, n+2, n+3 ... n+N+1 is used to provide the pixel data for a second display line. However, the data is used from the buffer such that all the pixel data for, for instance, pixel M is read for all the scan lines being used in the convolution before the pixel data for pixel M+1 is read from the buffer.

In some cases, the addressing and mapping scheme used for a frame buffer provides more than the data for a single pixel for each address. As illustrated in Figure 3 assume that a single address provides the pixel data for pixel 0 and pixel 1 of line 1, this is shown by the enclosing line 23. With the present invention, the data associated with line 23 is first read from the memory, followed by the convolution of data associated with lines 24 and 25. Then convolution is performed on the data for pixel 0 of lines 1, 2 and 3 for pixel 0, followed by the data for pixel 1, for lines 1, 2, 3. Now the data associated with lines 26, 27 and 28 is read from the memory, and so on.

In a specially organized frame buffer such as that described in a presently preferred embodiment, a single address provides the data for several lines in the buffer. For example, a single address may provide the data associated with lines 23, 24 and 25. In this event, the first data for pixel 0 is convolved, then that of pixel 1 is convolved. After that the data associated with lines 26, 27 and 28 is read from the memory and the data for pixel 2, and then 3 is convolved. This process continues for all the data along the line for scan lines 1, 2 and 3.

Thus, in general, the data for a first pixel for scan lines n, n+1, n+2 ... n+N is read from the buffer before the pixel data for subsequent pixels on these scan lines is read from the buffer. This data is then convolved to provide pixel data for a single pixel. This process is repeated for each of the pixels along the scan lines n, n+1, n+2 ... n+N. Following this the data for a first pixel along lines n+1, n+2... n+N+1 is read from the buffer, again before the pixel data for subsequent pixels along these lines. This data is then convolved to provide the filtered pixel data for the first pixel of the next display line. This process is repeated until vertical filtered data is provided for the entire display.

EMBODIMENTS OF THE CONVOLVER

As mentioned above, the pixel data trom "N+1" lines of data may be convolved. In the currently preferred embodiment N=2. (There is a discussion of convolution for other kernels later in this application.) In this case, N implements the equation :

$$\frac{aP_1+bP_2+aP_3}{2a+b}$$

where P_1 is the pixel data for the first pixel of the nth scan line, P2 the pixel data for the first pixel of the nth+1 line, and pixel 3 the pixel data for the n+2 scan line. "a"

and "b" are constants with "b" usually being greater than "a". In a typical application a=1 and b=2.

In Figure 4, the convolver (corresponding to kernel convolver 12 in Figure 1) includes a prescaler 32 which receives the input pixel data from the buffer. The amount of prescaling performed by prescaler 32 is controlled by the output of the coefficient table 33. The output of table 33 is controlled by the current cycle number which will be discussed. The output of the prescaler 32 provides one input to an adder 34. The other input to the adder 34, in effect is the output of the adder 34 after being coupled through the latch 35 and the multiplexer 31. The multiplexer 31 either provides as an input to the adder 34 the output of the latch 35 or the value 0. As will be seen at "cycle 0", the 0 input is provided to the adder 34, otherwise the contents of the latch 35 is the input to the adder 34. The contents of the latch 35 is normalized by a normalizer 36, the amount of normalization, typically a constant, is shown as normalization value 37. The output of the normalizer 36 is latched by latch 38, and the contents of this latch provide the pixel data for a pixel along a display line.

In practice, the prescaler is simply a digital shifter that provides digital multiplication by a factor of 1 or 2 and the normalizer 36 is another digital shifter which performs division by shifting the digital data by, for example, 2 places for division by four.

Assume first that a=1 and b=2 in the equation discussed above. Further assume that data is being scanned from a buffer in the manner illustrated and described in conjunction with Figure 2. The convolver can be seen to operate in a clock cycle sequence. During a cycle 0 the data associated with circle 16 is coupled to the prescaler 32. Cycle number 0 when applied to the coefficient table 33 causes the prescaler 32 to multiply this data by one, hence, the data is directly coupled to the adder 34. The cycle 0 coupled to the multiplexer 31 selects the zero input to the adder; therefore, 0 is added to the data associated with circle 16. This data is simply latched within latch 35 under control of the pixel clock. Next the data associated with circle 17 is coupled to the prescaler 32 on cycle 1. The cycle 1 input to the table 33 causes the prescaler to multiply this data by 2 (a left-shift of one) before coupling it to the adder 34. At the same time the output of the latch 35 is coupled through the multiplexer 31 and is added to the output of the prescaler 32. Hence, the sum P1 +2P2 is formed and coupled to the latch 35. Following this on cycle 2 the data associated with circle 18 is coupled to the prescaler 32. The cycle number "2" coupled to table 33 causes this data to be directly coupled to the adder 34. The adder 34 adds this data to the data contained within latch 35 forming the sum P1+2P2+P3. This sum is latched within latch 35 and then normalized by normalizer 36. For the described embodiment, normalizer 36 divides the data by a factor of 4 (a right-shift by 2) forming the final equation $\frac{P_1+P_2+P_3}{4}$. The resultant pixel data is latched in latch 38. On cycle 0 this data may be read from the latch 38 while new data for the next pixel is being coupled to the prescaler 32.

A fourth cycle may be used (i.e., cycle 3), in which event cycle 3 can control latch 38 with no data being shifted into the prescaler 32 during cycle 3. This can be used if 3 cycle timing is inconvenient.

An alternate convolver is illustrated in Figure 5. In this embodiment, an adder 40 receives as a first input, the output of the prescaler 43. Once again, the prescaler 43 receives the pixel data from the buffer. The amount of prescaling of prescaler 43 is controlled by the coefficient table 44. The output of table 44 is controlled by the cycle number coupled to the table. The other input ternunal of the adder 40 receives the output of the latch 42. The input to the latch is the output of the multiplexer 41. Multiplexer 41 selects either the output of the prescaler 43 or the output of the adder 40. The multiplexer 41 is controlled by the cycle 0 signal; for cycle 0 multiplexer 41 selects the output of the prescaler 43, otherwise it selects the output of the adder. The output of the latch 42 is coupled to a normalizer 46. The amount of normalization is controlled by the values shown as "normalization value 45". The output of the normalizer 45 is coupled to a latch 47. The output of the latch 47 provides the filtered pixel data. The circuit of Figure 5 performs the same convolution as the circuit of Figure 4.

Assume that the data for line n for pixel 0 is coupled to the prescaler 43. During the cycle 0 the multiplexer 41 selects the output of the prescaler 43 and couples the data into the latch 42. The prescaler 43 does not scale the data, because a=1 in the equation discussed above. The data for pixel 0 of line n+1 is prescaled by 2 and this data is then added to the contents of the latch with the sum being coupled to the multiplexer 41 and latched in latch 42. The process continues until the sum $aP1 +2P_2+aP_3$ is formed, computed and stored in latch 42. The normalizer 46 divides this sum by a factor of 4 and the resultant normalized value is coupled into the latch 47. Again, on cycle 0 (the start of new data into the prescaler 43 for the next pixel) the data is clocked from the latch thereby providing the filtered pixel data for the display. Once again, a four cycle scheme may be used with the fourth cycle (cycle 3) controlling latch 47.

In Figure 7A a general prescaler is shown comprising a multiplier 50. The input pixel data is coupled to the multiplier, the output of the multiplier provides the scaled pixel data. The amount of multiplication is controlled by the output of the coefficient lookup table 51. This output is determined by the cycle number. The cycle number (e.g., 1,2, 3 . . .) selects the amount of multiplication required for the convolution being used and thereby controls the amount of multiplication performed by the multiplier 50.

Figure 7B illustrates a prescaler which may be used when the multiplication used by the convolution step involves multiplication by one or by two. In this case a multiplexer 53 receives the input pixel data at one terminal and the input pixel data multiplied by two (i.e., left-shift-

ed by 1 with a zero filling in on the right) at its other terminal. The cycle number requiring the "x2" pixel data is used to select the "0" input to the multiplexer 53 and thus provides the needed scaled input pixel data.

The convolvers discussed above are particularly good for a serial kernel data stream. Figure 6 illustrates a convolver implemented in a table 71 which can be used for a parallel data stream. It is particularly useful when a limited number of bits are used; for example, in a 1 bit/pixel display where the 1-2-1 kernel is used. The results of the convolution arithmetic are precomputed and placed in the table. This is used as will be seen for software embodiments of the invention where the color lookup table is preloaded for use as a convolution lookup table.

SOFTWARE EMBODIMENT OF THE PRESENT INVENTION

The method of the present invention can be readily implemented in software to provide real time convolution. An embodiment of the invention is described below for a "chunky" frame buffer.

With a chunky frame buffer, all the bits for a given pixel are stored as adjacent bits of a memory word. For example, if color depth is 4 bits per pixel, and the CPU word size is 32 bits, then 8 pixels are stored in each CPU word. Unlike the planar frame buffer, a given CPU access will always access all the bits in a given pixels, and in some cases, the bits in adjacent pixels. Chunky frame buffers are also used in commercially available computers such as Apple Computer, Inc.'s Macintosh II computer.

SOFTWARE EMBODIMENT FOR THE CHUNKY FRAME BUFFER

In this embodiment, a one bit per pixel real-time convolution with a chunky frame buffer is realized. Unlike a method for a planar frame buffer, the exact number of bits per pixel cannot be obtained when rearranging the data, hence, the next power of 2 greater than the number of bits needed is used. For the described embodiment, three-lines are used for the convolution and hence, four bits of pixel data are stored in a buffer for each pixel. The method described below places the bits in their proper position.

First, it should be noted that a one bit per pixel frame buffer "off screen" in RAM is used by the CPU for drawing. This frame buffer is separate from the four bit per pixel frame buffer that is actually scanned to provide the display. The method described below reads data from the one bit per pixel frame buffer, expands the data to the four bits per pixel, then writes the data into the four bit per pixel frame buffer. The method merges the pixel information from the two previous lines before it writes the results into the four bit per pixel frame buffer. When the four bit pixel is presented to the color lookup table, the three bits for lines n-1, n and n+1 are available to lookup the proper gray scale for the 1-2-1 convolution. Again, as with the previous embodiment, the color lookup table is loaded with gray scale information to provide the convolution. (Three of the four bits read from the four bit per pixel frame buffer are used by the CLUT to provide the output convolved signal for the display.)

Step 0

Four 32-bit words (A, B, C, and D) are initialized to zero. (A, B, C, and D each refer to 32-bit registers within the CPU.) A 32-bit word R is read starting from the left-most pixel position of the top scan line of the one bit per pixel frame buffer. A, B, C and D are all stored at adjacent left to right locations starting from the top scan line of the four bit per pixel frame buffer.

Step 1

R is read from the next 32 bits in the one bit per pixel frame buffer immediately below the last 32-bit word read from the one bit per pixel frame buffer. This is shown in Figure 8 where two words, words 93 and 94, are shown for lines n and n+1 in the one bit per pixel frame buffer.

Step 2

As shown in Figure 9, one byte of R is expanded into a second 32-bit word M such that each of the 8 bits is placed at 4 bit intervals in the 32-bit word starting at bit 1 (i.e., bit 0 to bit 1, bit 1 to bit 5, bit 2 to bit 9, etc.) and a 1 is placed in every 4th bit starting at bit 0. All other bits are set to zero. For example, the byte 0111 0101 is converted to (shown as groups of 4): 0001 0011 0011 0011 0001 0011 0001 0011. This is done by using a 256x32-bit pre-loaded lookup table in RAM.

Step 3

A is left-shifted by 1. In some microprocessors such as the Motorola Part No. 68020 this can be accomplished more quickly by adding A to itself. In the upper part of Figure 10, A is shown before the shift and in the lower part of Figure 10 after the shift.

Step 4

M is bit-wise ORed into A as shown in Figure 11. First, this serves to merge the byte from R into A since it is known that the bits in A corresponding to the bits from the byte from R are all zero (anything ORed with zero retains its value). Second, this serves to force every 4th bit starting with bit 0 in A to one (this sets up for the merge operation in step 10, below).

Step 5

A is stored in the four bit per pixel frame buffer immediately below the last place A was stored as shown in Figure 12.

Step 6

Steps 2 through 4 are repeated for the three other byres from R. This time, however, B, C, and D are used instead of A.

Step 7

R is read for the next 32-bit word in the one bit per pixel frame buffer immediately below the last 32-bit word as in Step 1 above.

Step 8

As shown in Figure 9, one byte of R is expanded into M with each of the eight bits placed at 4 bit intervals starting at bit 1. Also, a 0 is placed in every 4th bit starting at bit 0 and all other bits are set to 1 . For example, the byte 0111 0101 would be converted to 1100 1110 1110 1110 1100 1110 1100 1110. This is accomplished by means of a second 256x32-bit pre-loaded lookup table in RAM.

Step 9

As shown in Figure 10, A is left-shifted by 1. Once again, as mentioned for step 3, addition of A to itself may be used.

Step 10

As shown in Figure 11, M is bit-wise ANDed into A. First, this serves to merge the byte from R into A since it is known that the bits in A corresponding to the bits from the byte from R are all ones (anything ANDed with one retains its value). Second, this serves to force every 4th bit starting with bit 0 in A to zero (this will set up the merge operation in step 4, above).

Step 11

A is stored in the 4-bit frame buffer immediately below the last place A was stored. See word 95 of Figure 12.

Step 12

Steps 8 through 10 are repeated for the 3 other bytes from R. They are merged in B, C, and D instead of A. See words 96, 97 and 98 of Figure 12.

Step 13

Steps 1 through 12 are repeated until the bottom of the frame buffer is reached, then R is read for the pixels on the top scan line of the 1 bit/pixel frame buffer just to the right of where it was loaded at the start of the last pass. A, B, C, and D are all stored at adjacent left-to-right locations on the top scan line of the 4 bit/pixel frame buffer just -to the right of where they were loaded at the start of the last pass.

In summary, the pixels in the 4-bit per pixel frame buffer 100 of Figure 12 are coded with line n+1 in bit 1, n in bit 2, and n-1 in bit 3 (this resulting bit configuration is shown in Figure 11). Bit 0 is ignored by the CLUT 101 of Figure 12. The one bit per pixel frame buffer of Figure 8 is scanned vertically with a new bit added into each four bit pixel for each scan line by left shifting the existing bit for the pixel by one and merging the new bit into bit 1 of the 4-bit per pixel word. The shift operation serves to adjust the pixel from its previous centering on line n-1 (the line above) to its current centering on line n. In other words, when the operation begins the four bit pixel data contains bits from lines n-2, n-1 and n since the data was used for the line above. The left shift operation changes the configuration of the four bits to n-1, n, and a one or a zero in bit 1 (bit 0 is ignored). Then, the new bit from the one bit per pixel frame buffer is merged into bit one for line n+1. The new assembled four bit word is stored in the four bit per pixel frame buffer and as mentioned, the CLUT is used to provide the convolution.

In detail, the method starts in the upper-left of the frame buffer and works down a 32-pixel column. The read into R loads the 32 1 bit pixels then each 8 pixels of the 32 are operated upon separately. The first 8 pixels (a byte) are used as a lookup table index to fetch a 32-bit word, M. M holds the 8 pixels, spread out at 4-bit intervals so that they are ready to merge for the 4 bit/pixel frame buffer.

M also is set up with the rest of its bits prepared for either a bit-wise AND merge or an OR merge. The reason it alternates between AND and OR is that it saves the step of clearing (or setting) the bits in A which are the destination for the 8 pixels from R. Since A will be left-shifted just prior to the AND or OR merge, the bit immediately to the right of the destination of the R bits is forced so that at the next step they are already prepared for merging. AND prepares for the OR by forcing zeroes, and OR prepares for the AND by forcing ones.

A is left-shifted by one to update the pixel from being centered for the previous line to being centered for the current line. The left-shift moves the previous line n+1 to the current line n and the previous line n to the current line n-1. Previous line n-1 (current line n-2) is shifted out. Notice that this shift applies to all eight pixels contained in the 32 bits of A so it is an 8-way parallel operation. Notice also the bits from previous line n-1 shifts into the unused bit of the next 4-bit pixel to the left (or off the left edge of the 32-bit word).

Then, M is merged with A by either an AND or an OR. Bits from n and n-1 are left alone, new n+1 bits are merged in, and the unused bits are set to known state (0 if an AND, 1 if an OR). A is finally stored in the 4 bit/pixel frame buffer.

The other 24 pixels in R are handled the same way, with 8 pixels each for B, C, and D.

The same steps are performed for each successive scan line below until the bottom of the frame buffer is reached. Then, the next column of 32 pixels immediately to the right is scanned-down, and so on until the entire frame is scanned.

The CLUT 101 of Figure 12 is loaded in a similar manner to that of a planar frame buffer implementation shown in Figure 13. The differences are that the bit ordering is different and that since bit 0 in the 4-bit pixels is indeterminate (it alternates depending on whether the last merge was with an AND or an OR), the same gray value for every two CLUT entries is stored.

OTHER CONVOLUTION KERNELS

In the previous section, most of the emphasis has been on the 1-2-1 kernel. Experiments have shown that neither a 3-line convolution nor on-off-on-off reduction of 50% gray is essential in all situations for effective interlace flicker reduction. It the constraint that on-off-on-off horizontal line patterns are reduced to a 50% gray is maintained and other kernel sizes are tried other than 1 x 3, for each kernel size there is one set of coefficients to meet the on-off-on-off constraint. These coefficients match Pasqual's triangle (i.e.,1; 1, 1; 1, 2, 1; 1, 3, 3, 1; 1, 4, 6, 4, 1; etc.).

ADAPTIVE CONVOLUTION

As mentioned above, the worst flicker patterns are the ones which have high horizontal coherence (i.e., repeat horizontally). Horizontal solid lines, horizontal dashed lines, and gray dither patterns are examples of patterns with high horizontal coherence. Text is an example of patterns without such coherence. The convolution discussed above may be adaptive, that is, it may be varied depending on the type of patterns being displayed. First, it is determined whether a repeating pattern is occurring in a local horizontal group ot kernels, for example, 8 pixels across. If there is a pattern in the kernels, then all of the kernels are convolved for example, with the 1-2-1 coefficients. If there is no such pattern, then the 8 pixels are convolved with coefficients making a sharper filter (e.g., 1-3-1 or 1-4-1). The test to determine whether a pattern is repeating must be applied continuously in a moving horizontal window, kernel by kernel. Since the test windows overlap, some kernels may be part of a pattern in one test window but not in another. For these kernels, the 1-2-1 convolution is used, since they are at the edge of the patern. Different tests may be used for determining whether a pattern is being repeated, for example, the left four kernels may be compared with the right four kernels within the window.

Claims

1. A method for providing filtered pixel data for a display stored in a raster-scanned video graphics apparatus having a first memory where pixel data for each scan line is stored in adjacent locations in said first memory such that each word accessed from said first memory includes the data for at least one pixel, and a second memory storing first coded pixel data of n-1, n, and n+1 scan lines in adjacent locations, where n represents a given scan line in the first memory, said method comprising the steps of:

   reading pixel data from scan line n+2 from the first memory;
   shifting the first coded pixel data of the second memory;
   merging the pixel data from scan line n+2 with the shifted first coded pixel data to generate second coded pixel data in the second memory for n, n+1, and n+2 scan lines; and
   convolving said second coded pixel data to generate said filtered pixel data.

2. The method defined by Claim 1 wherein said convolving step comprises the step of performing the following computer implemented computation:

$$\frac{aP_1+bP_2+aP_3}{2a+b,}$$

   where P_1 is the pixel data for the nth scan line, P_2 is the pixel data for scan line n + 1, and P_3 is the pixel data for scan line n + 2, and wherein a and b are constants.

3. The method as defined by Claim 1 wherein the step of convolving comprises the further step of generating an RGB output signal based on the pixel data.

4. The method as defined by Claim 3 further comprising the step of applying the RGB output signal to a color monitor to display a representation of the pixel data.

Patentansprüche

1. Verfahren zum Bereitstellen gefilterter Bildpunktdaten für einen Bildschirm, die in einer Rasterabtast-Videographik-Vorrichtung abgelegt sind, die einen ersten Speicher hat, wobei Bildpunktdaten für jede

Bildzeile in benachbarten Speicherplätzen in dem ersten Speicher abgelegt werden, so daß von dem ersten Speicher jedes Wort, auf das zugegriffen wird, die Daten für mindestens einen Bildpunkt enthält, und einen zweiten Speicher, der erste kodierte Bildpunktdaten von Bildzeilen n-1, n und n+1 in benachbarten Speicherplätzen ablegt, wobei n für eine bestimmte Bildzeile in dem ersten Speicher steht, wobei das Verfahren die Schritte beinhaltet:

Lesen von Bildpunktdaten aus Bildzeile n+2 aus dem ersten Speicher;
Verschieben der ersten kodierten Bildpunktdaten des zweiten Speichers;
Vereinigen der Bildpunktdaten aus Bildzeile n+2 mit den verschobenen ersten kodierten Bildpunktdaten, um zweite kodierte Bildpunktdaten in dem zweiten Speicher für Bildzeilen n, n+1 und n+2 zu erzeugen; und
Verknüpfen der zweiten kodierten Bildpunktdaten, um die gefilterten Bildpunktdaten zu erzeugen.

2. Verfahren nach Anspruch 1, bei dem der Verknüpfungsschritt den Schritt umfaßt, die folgende computer-implementierte Rechnung durchzuführen:

$$\frac{aP_1+bP_2+aP_3}{2a+b,}$$

wobei P_1 das Bildpunktdatum der n-ten Bildzeile ist, P_2 das Bildpunktdatum der Bildzeile n+1 ist, und P_3 ist das Bildpunktdatum der Bildzeile n+2, und wobei a und b Konstanten sind.

3. Verfahren nach Anspruch 1, bei dem der Verknüpfungsschritt ferner den Schritt umfaßt, daß ein auf den Bildpunktdaten basierendes RGB-Ausgangssignal generiert wird.

4. Verfahren nach Anspruch 3, ferner den Schritt beinhaltend, daß das RGB-Ausgangssignal einem Farbbildschirm zugeführt wird, um eine Darstellung der Bildpunktdaten zu zeigen.

Revendications

1. Un procédé de génération de données filtrées de pixels pour une affichage mémorisé dans un appareil graphique vidéo à balayage de trame qui inclut une première mémoire, des données de pixels pour chaque ligne de balayage étant mémorisées dans des emplacements adjacents de ladite première mémoire d'une manière telle que chaque mot auquel un accès est effectué à partir de ladite première mémoire inclut les données pour au moins un pixel, et une deuxième mémoire qui mémorise des premières données codées de pixels de lignes de balayage n - 1, n , n + 1 dans des emplacements adjacents, où n représente une ligne de balayage donnée de la première mémoire, ledit procédé comprenant les étapes consistant à:

lire, dans la première mémoire, des données de pixels provenant de la ligne n + 2;
décaler les premières données codées de pixels de la deuxième mémoire;
fusionner les données de pixels provenant de la ligne de balayage n + 2 avec les premières données codées décalées de pixels pour engendrer des deuxièmes données codées de pixels dans la deuxième mémoire pour des lignes de balayage n, n + 1, n + 2; et
effectuer une convolution desdites deuxièmes données codées pour engendrer lesdites données filtrées de pixels.

2. Le procédé selon la revendication 1, dans lequel ladite étape de convolution comprend l'étape consistant à effectuer le calcul suivant:

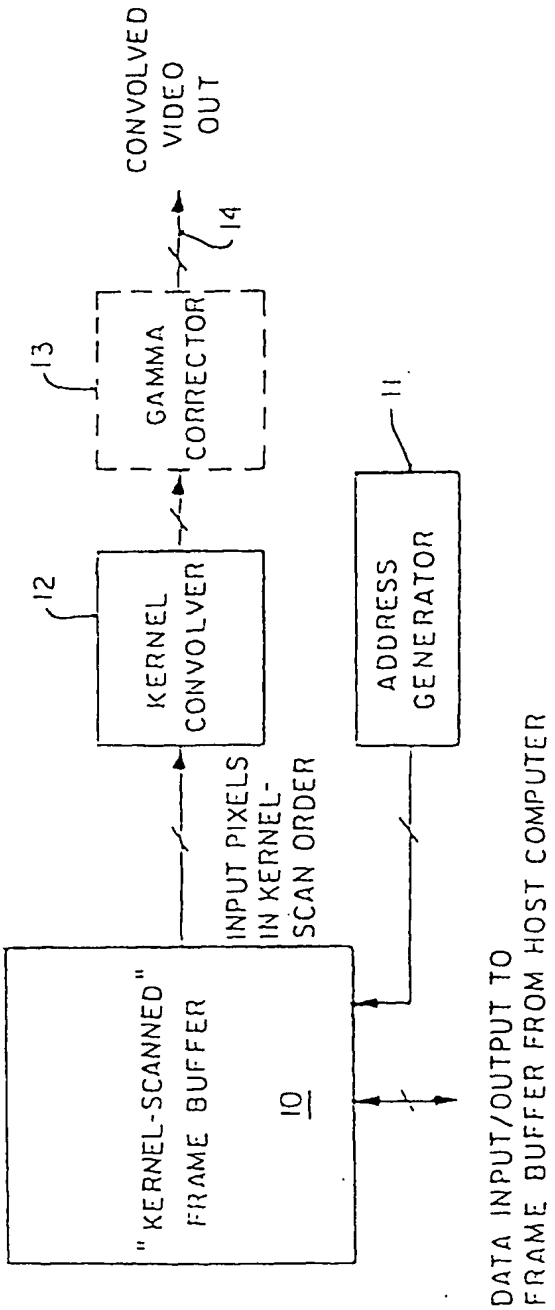
$$\frac{a\,P_1 + b\,P_2 + a\,P_3}{2a + b}$$

où P_1 est la donnée de pixel pour la n-ième ligne de balayage, P_2 la donnée de pixel pour la ligne de balayage n + 1, et P_3 la donnée de pixel pour la ligne de balayage n + 2; et où a et b sont des constantes.

3. Le procédé selon la revendication 1 dans lequel ladite étape de convolution comprend l'étape additionnelle consistant à engendrer un signal de sortie RGB sur la base des données de pixels.

4. Le procédé selon la revendication 3 qui comprend en outre l'étape consistant à appliquer le signal de sortie RGB à un monitor en couleurs pour afficher une représentation des données de pixels.

FIG. 1


"KERNEL-SCANNED" FRAME BUFFER
10
INPUT PIXELS IN KERNEL-SCAN ORDER
KERNEL CONVOLVER
12
GAMMA CORRECTOR
13
14
CONVOLVED VIDEO OUT
ADDRESS GENERATOR
11
DATA INPUT/OUTPUT TO FRAME BUFFER FROM HOST COMPUTER

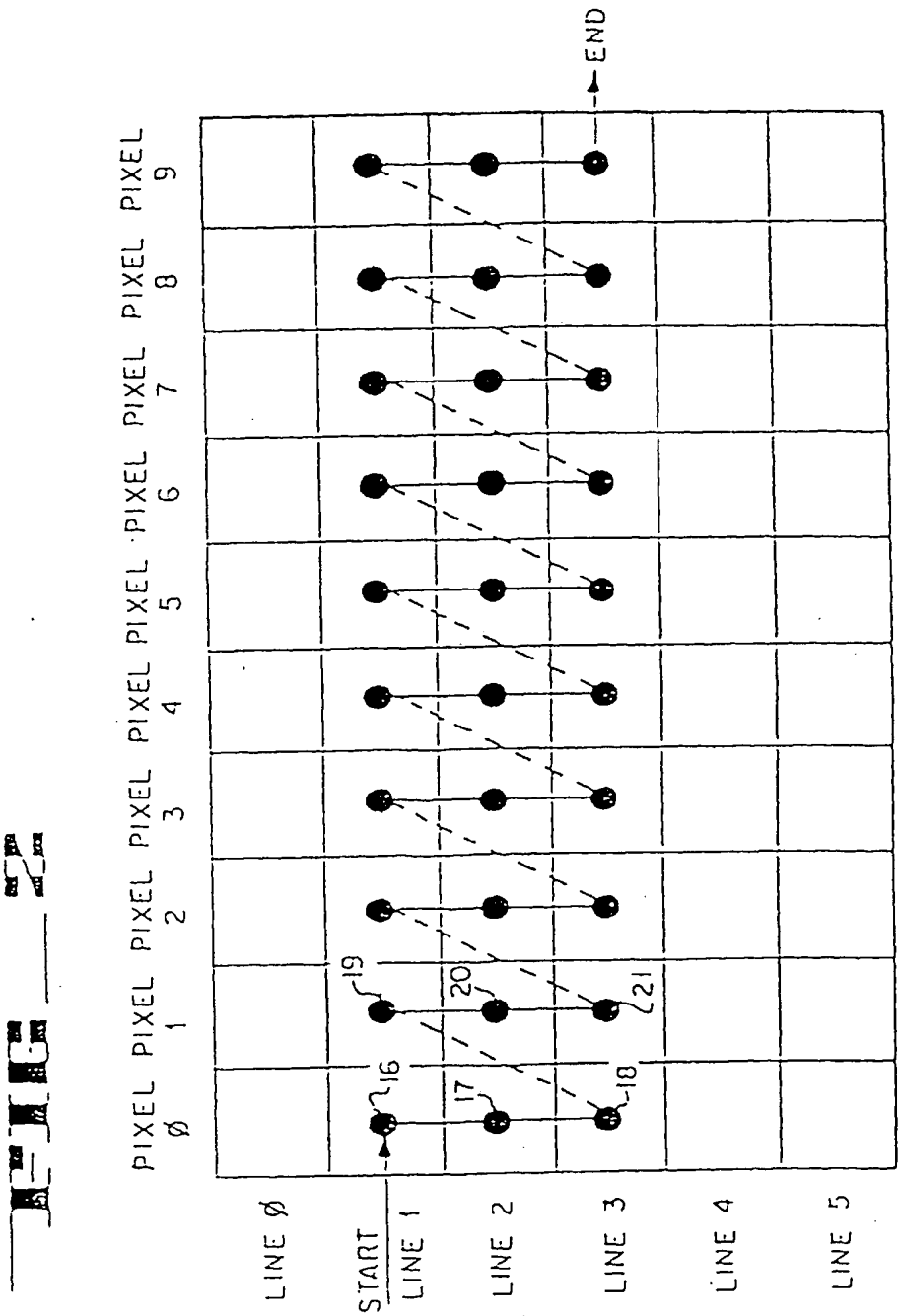
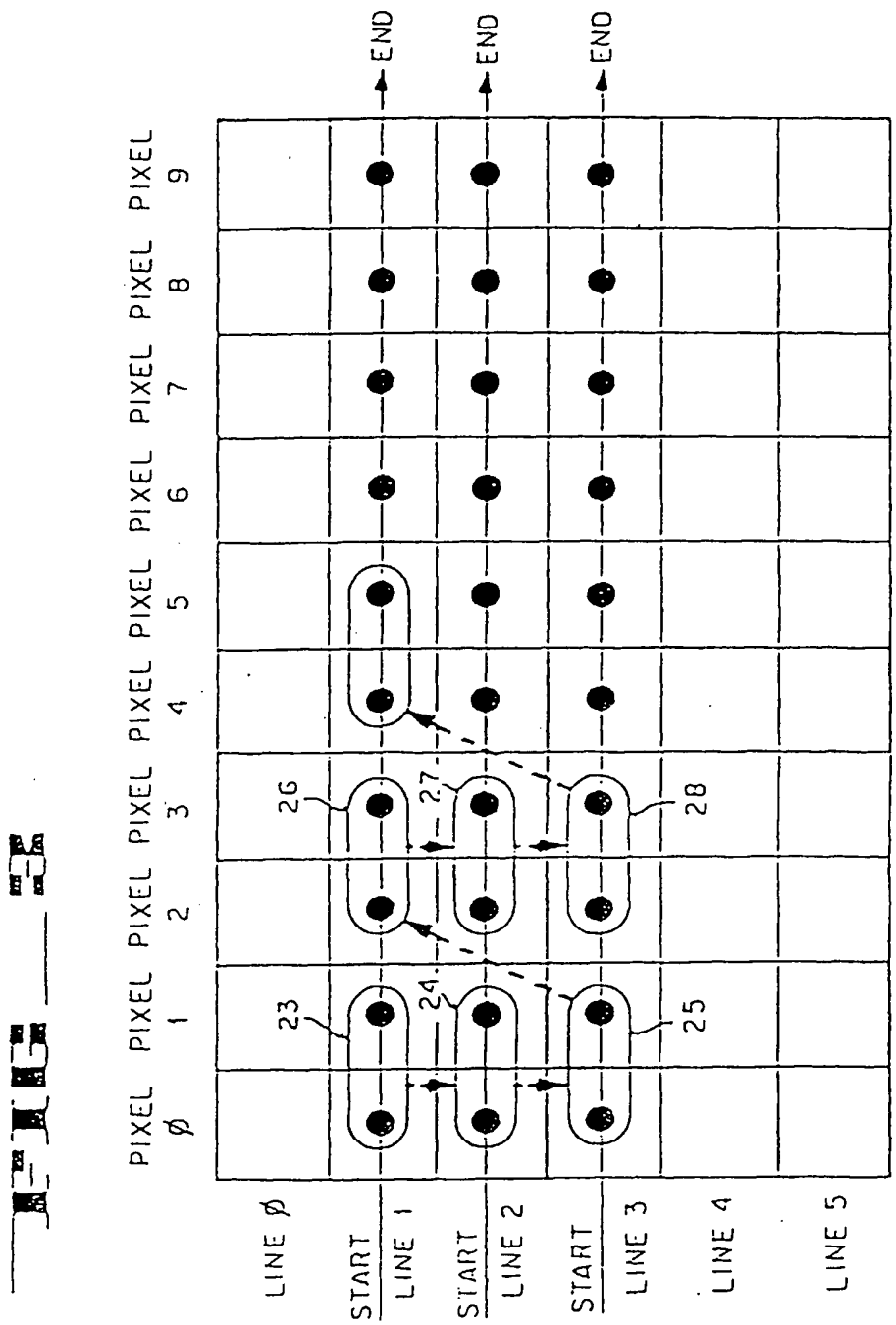

PIXEL Ø
PIXEL 1
PIXEL 2
PIXEL 3
PIXEL 4
PIXEL 5
PIXEL 6
PIXEL 7
PIXEL 8
PIXEL 9
LINE Ø
START
LINE 1
LINE 2
LINE 3
LINE 4
LINE 5
END
16
17
18
19
20
21

FIG. 2


PIXEL Ø
PIXEL 1
PIXEL 2
PIXEL 3
PIXEL 4
PIXEL 5
PIXEL 6
PIXEL 7
PIXEL 8
PIXEL 9
END
END
END
LINE Ø
START
LINE 1
START
LINE 2
START
LINE 3
LINE 4
LINE 5
23
24
25
26
27
28

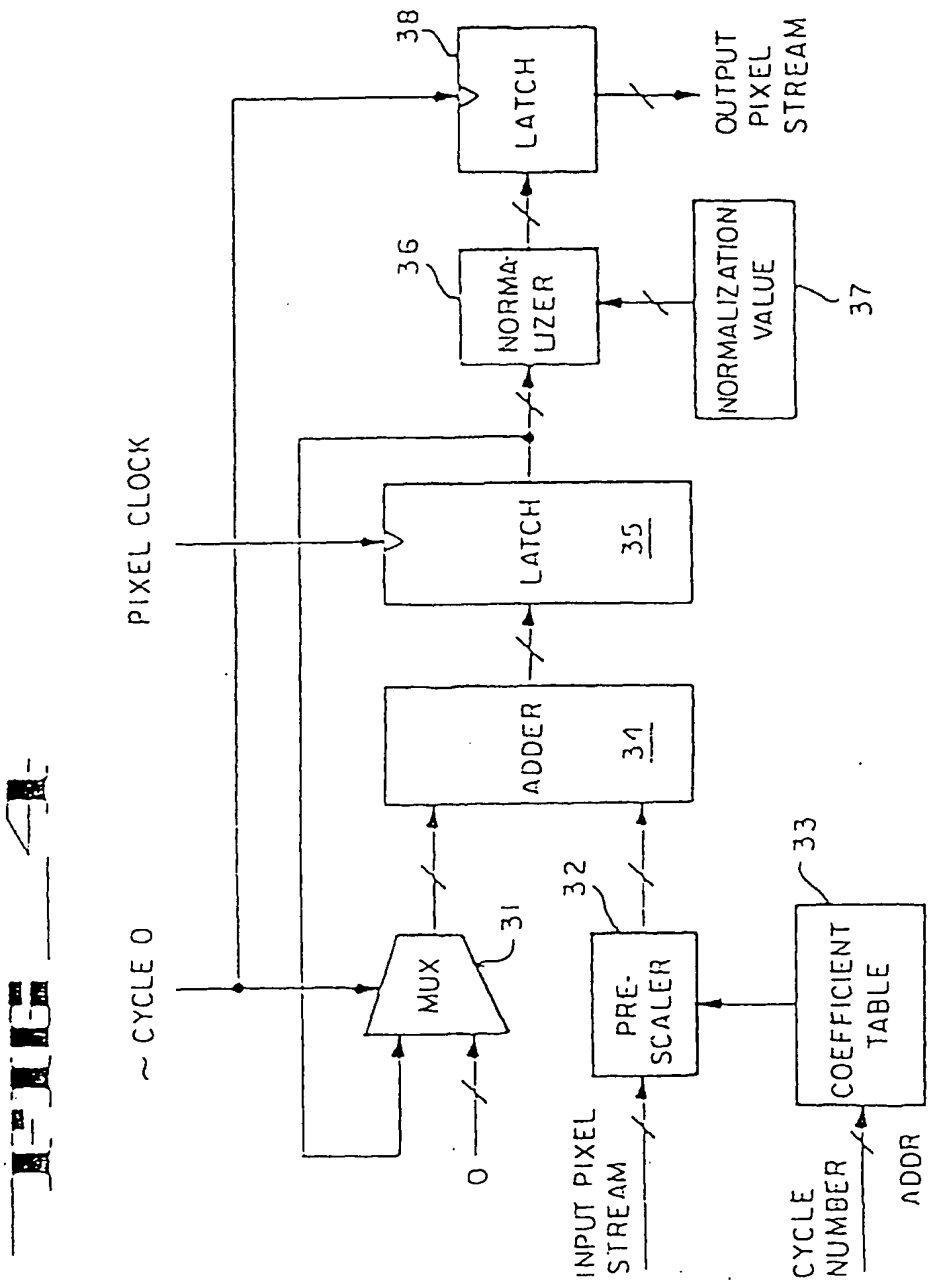
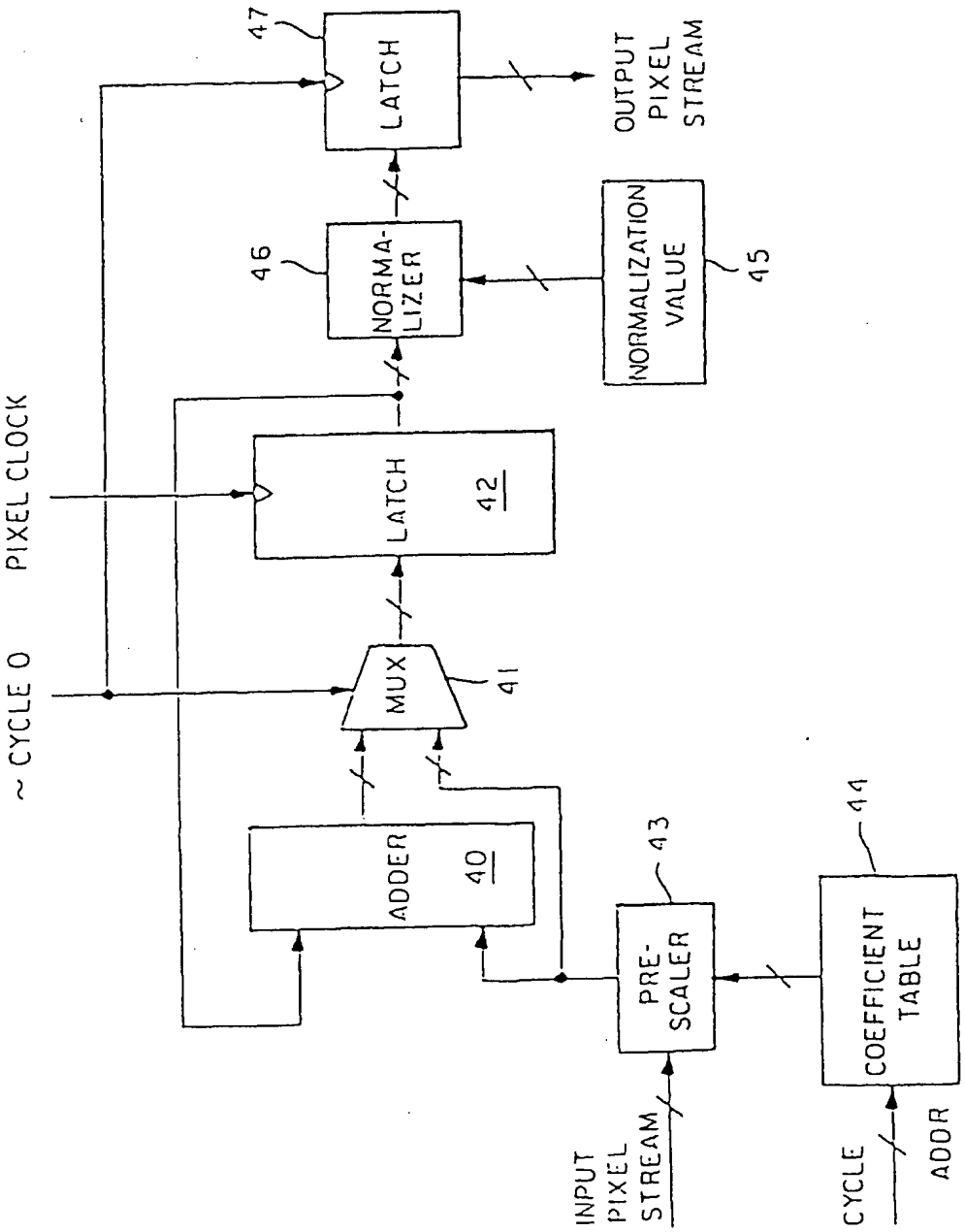

38
LATCH
OUTPUT PIXEL STREAM
36
NORMA-LIZER
NORMALIZATION VALUE
37
PIXEL CLOCK
LATCH
35
ADDER
31
~ CYCLE 0
MUX
31
32
PRE-SCALER
33
COEFFICIENT TABLE
0
INPUT PIXEL STREAM
CYCLE NUMBER
ADDR

FIG. 5


~ CYCLE 0
PIXEL CLOCK
MUX
41
LATCH
42
ADDER
40
PRE-
SCALER
43
COEFFICIENT
TABLE
44
INPUT
PIXEL
STREAM
CYCLE
ADDR
NORMA-
LIZER
46
NORMALIZATION
VALUE
45
LATCH
47
OUTPUT
PIXEL
STREAM

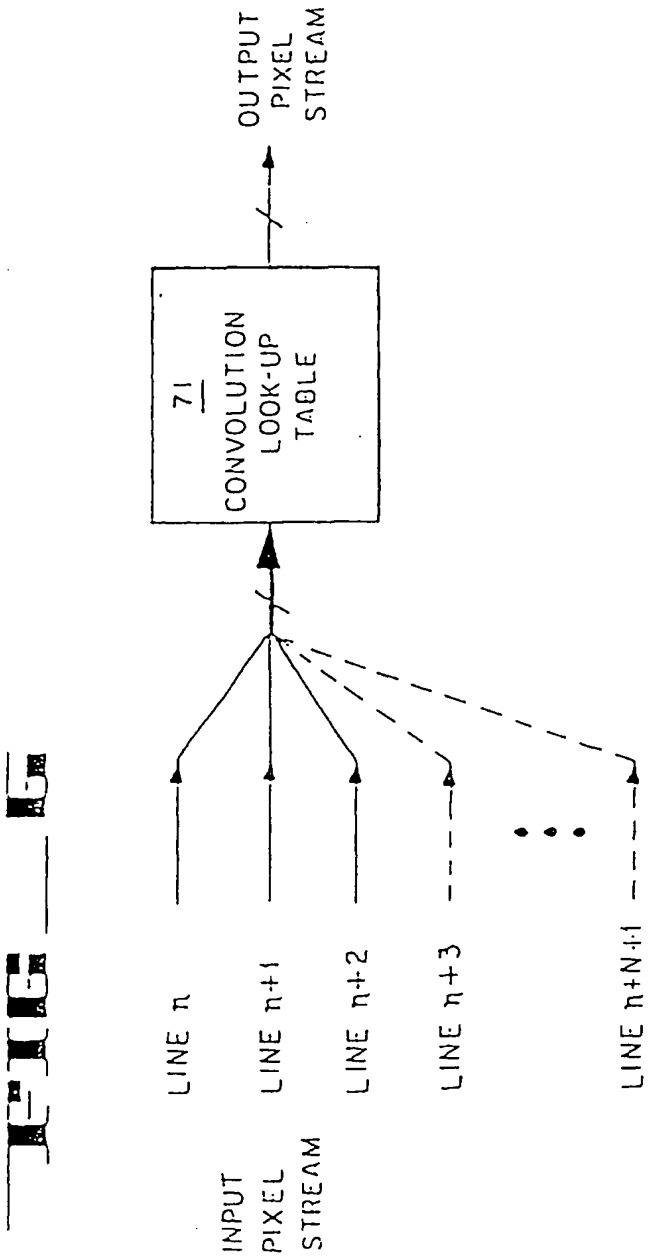

OUTPUT PIXEL STREAM
71
CONVOLUTION LOOK-UP TABLE
LINE n
LINE n+1
LINE n+2
LINE n+3
LINE n+N-1
INPUT PIXEL STREAM

FIG. 6

FIG 7A

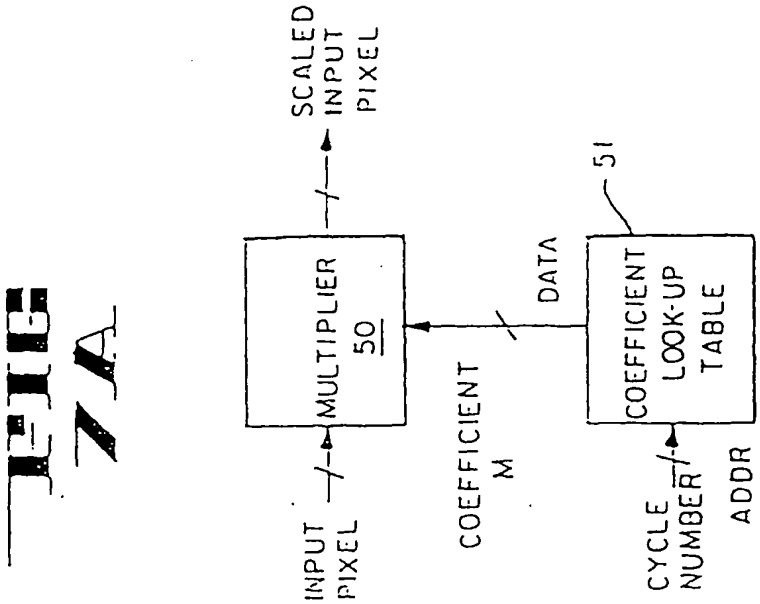

SCALED INPUT PIXEL
MULTIPLIER 50
INPUT PIXEL
COEFFICIENT M
DATA
51
COEFFICIENT LOOK-UP TABLE
CYCLE NUMBER
ADDR

FIG 7B

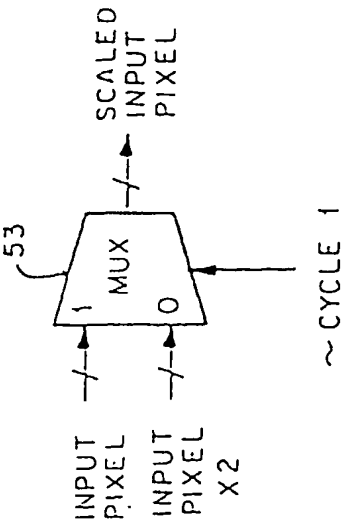
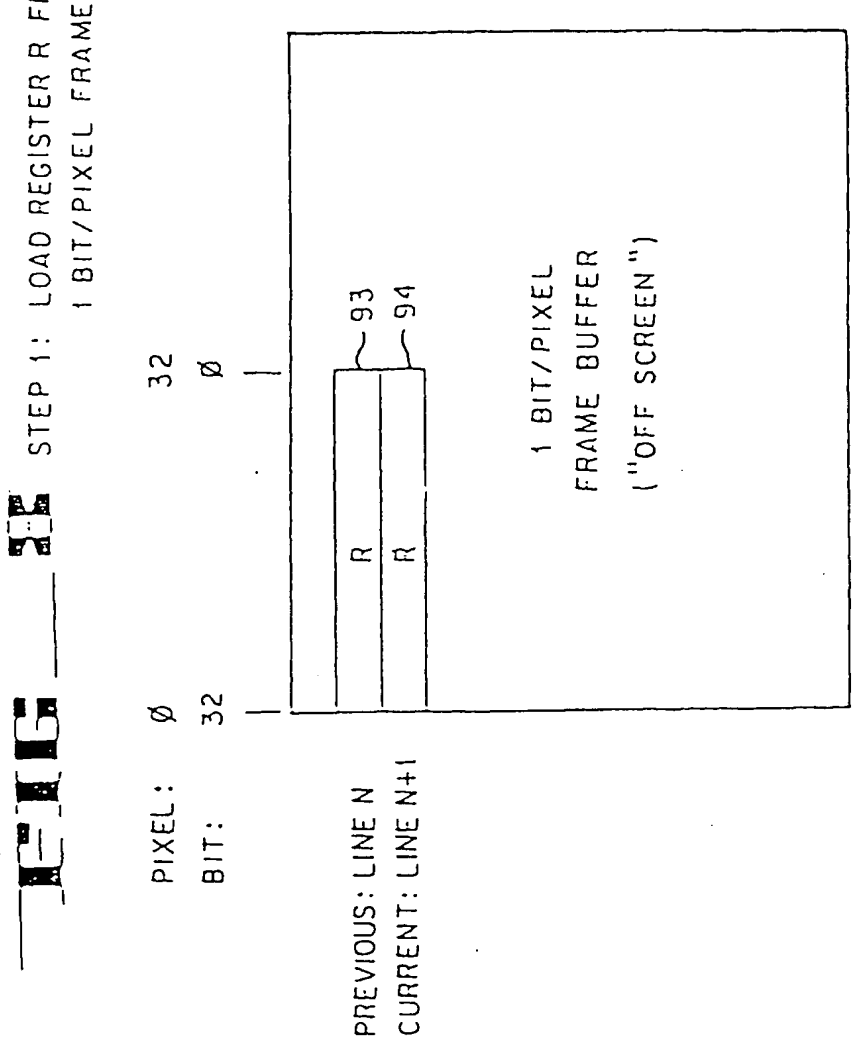

SCALED INPUT PIXEL
53
MUX
1
0
INPUT PIXEL
INPUT PIXEL X2
~CYCLE 1

FIG. 8B STEP 1: LOAD REGISTER R FROM THE 1 BIT/PIXEL FRAME BUFFER



(WORD AND LINE SIZE RELATIVE TO FRAME BUFFER SIZE NOT SHOWN TO SCALE)

FIG. 4A

STEP 2: LOAD M WITH TABLE LOOK-UP OF FIRST BYTE OF R

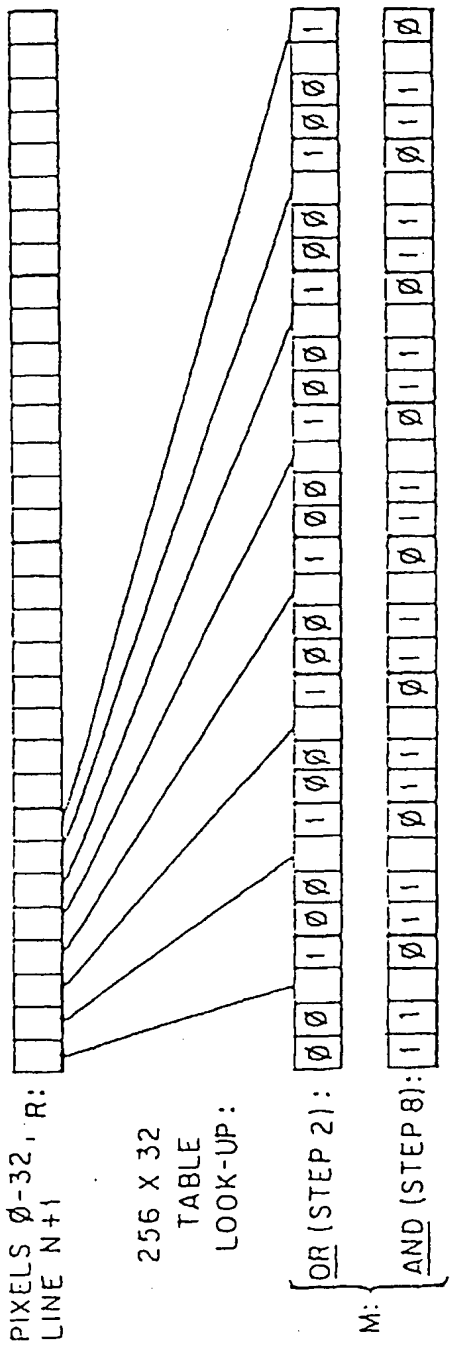


FIG. 10A

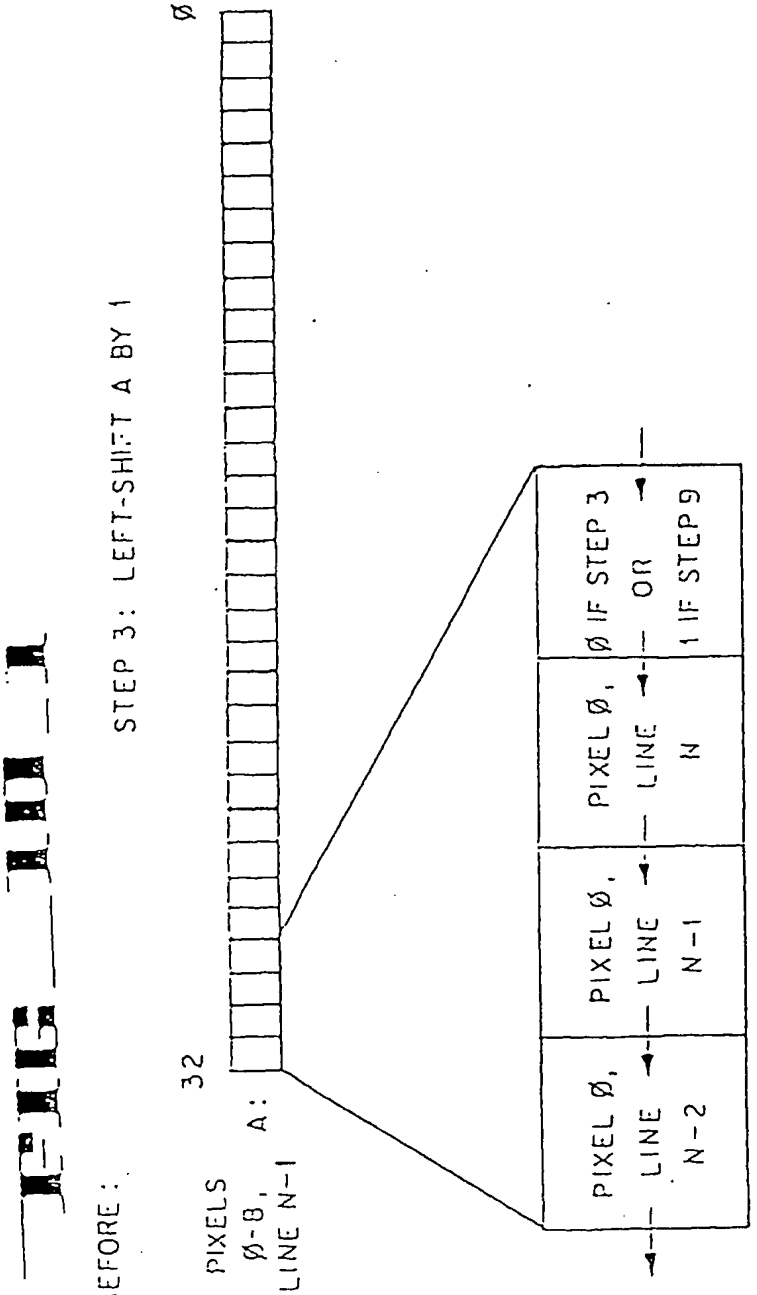
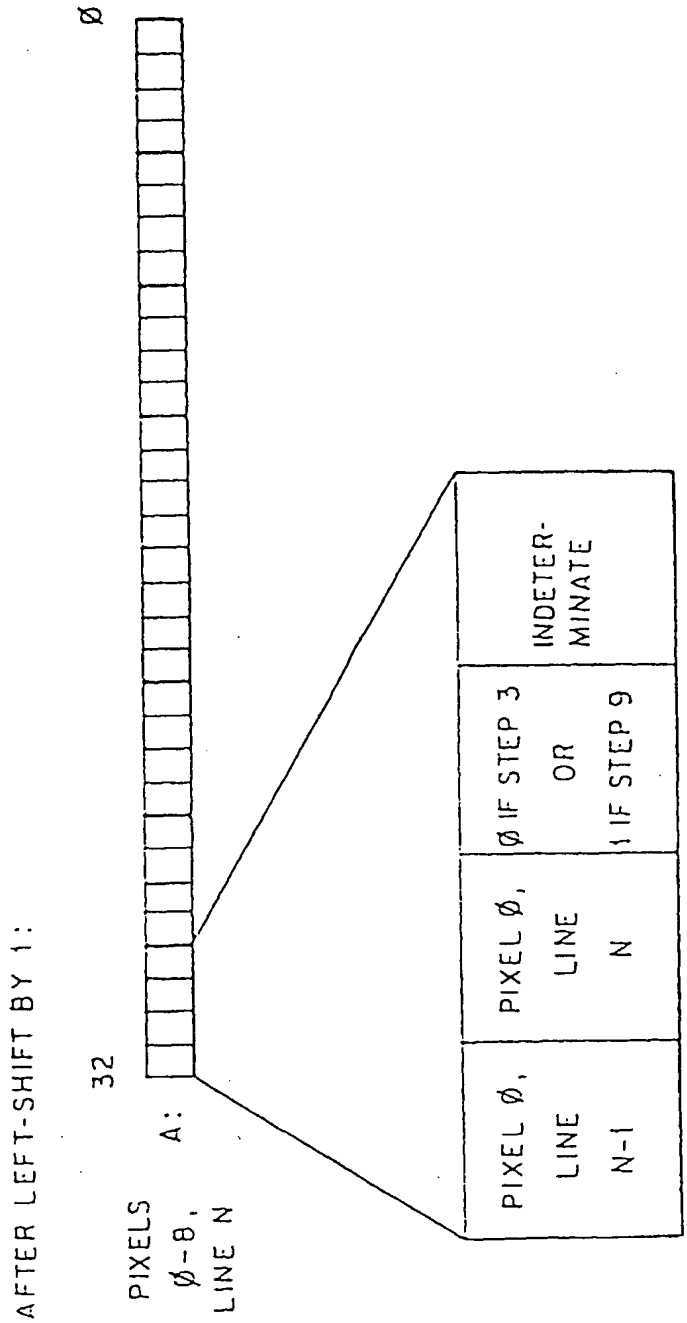

BEFORE:
PIXELS Ø-8, LINE N-1
A:
32
Ø
STEP 3: LEFT-SHIFT A BY 1
PIXEL Ø, LINE N-2
PIXEL Ø, LINE N-1
PIXEL Ø, LINE N
Ø IF STEP 3 OR 1 IF STEP 9
SIMILARLY,
B HOLDS PIXELS 9-15
C HOLDS PIXELS 16-23
D HOLDS PIXELS 24-31

FIG_10_2


AFTER LEFT-SHIFT BY 1:
PIXELS Ø-8, LINE N
A:
32
Ø
PIXEL Ø, LINE N-1
PIXEL Ø, LINE N
Ø IF STEP 3 OR 1 IF STEP 9
INDETER-MINATE

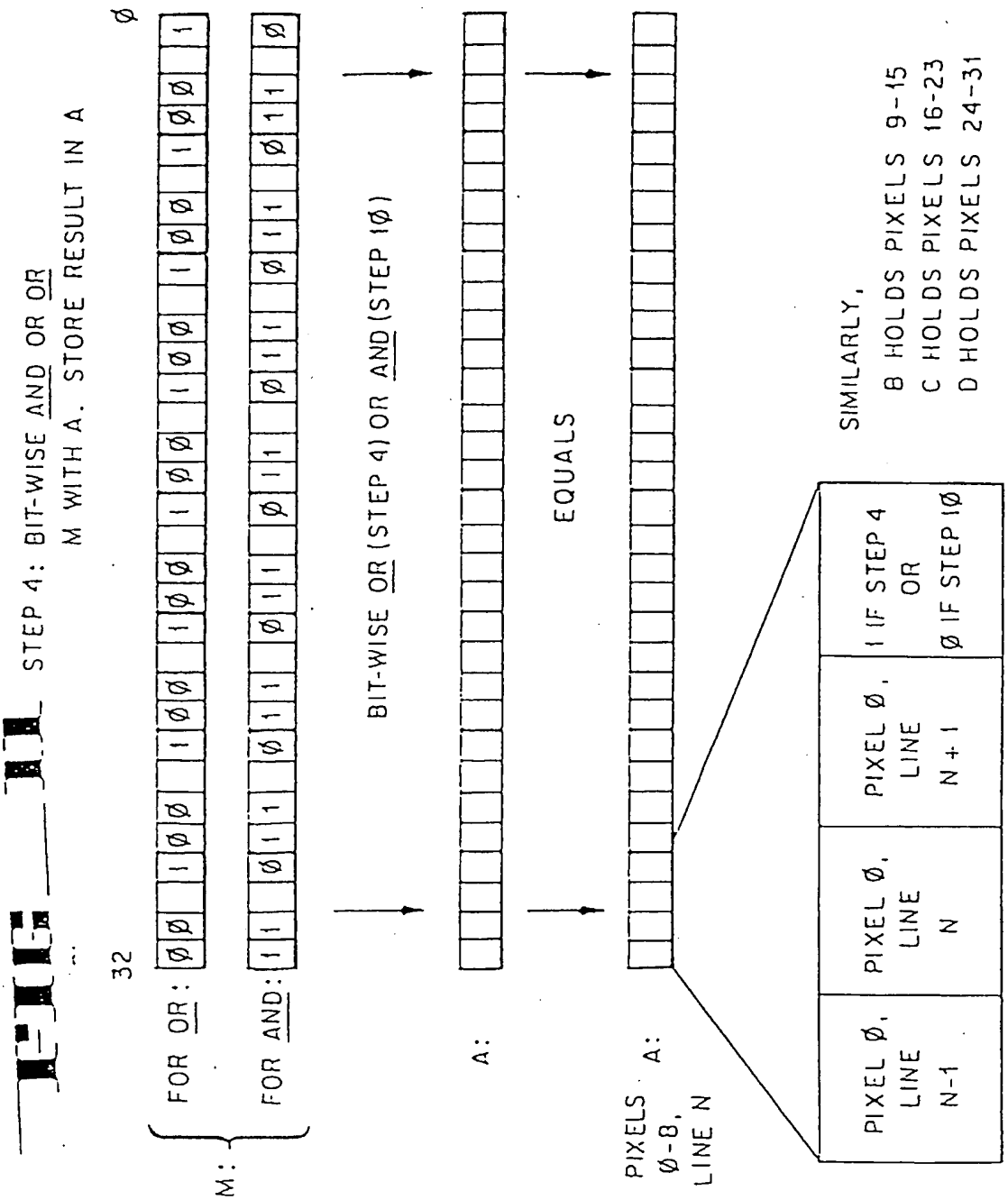


FIG. 11
STEP 4: BIT-WISE AND OR OR
M WITH A. STORE RESULT IN A
32
Ø
M:
FOR OR:
FOR AND:
BIT-WISE OR (STEP 4) OR AND (STEP 1Ø)
A:
EQUALS
PIXELS Ø-8, LINE N
A:
PIXEL Ø, LINE N-1
PIXEL Ø, LINE N
PIXEL Ø, LINE N+1
1 IF STEP 4 OR Ø IF STEP 1Ø
SIMILARLY,
B HOLDS PIXELS 9-15
C HOLDS PIXELS 16-23
D HOLDS PIXELS 24-31

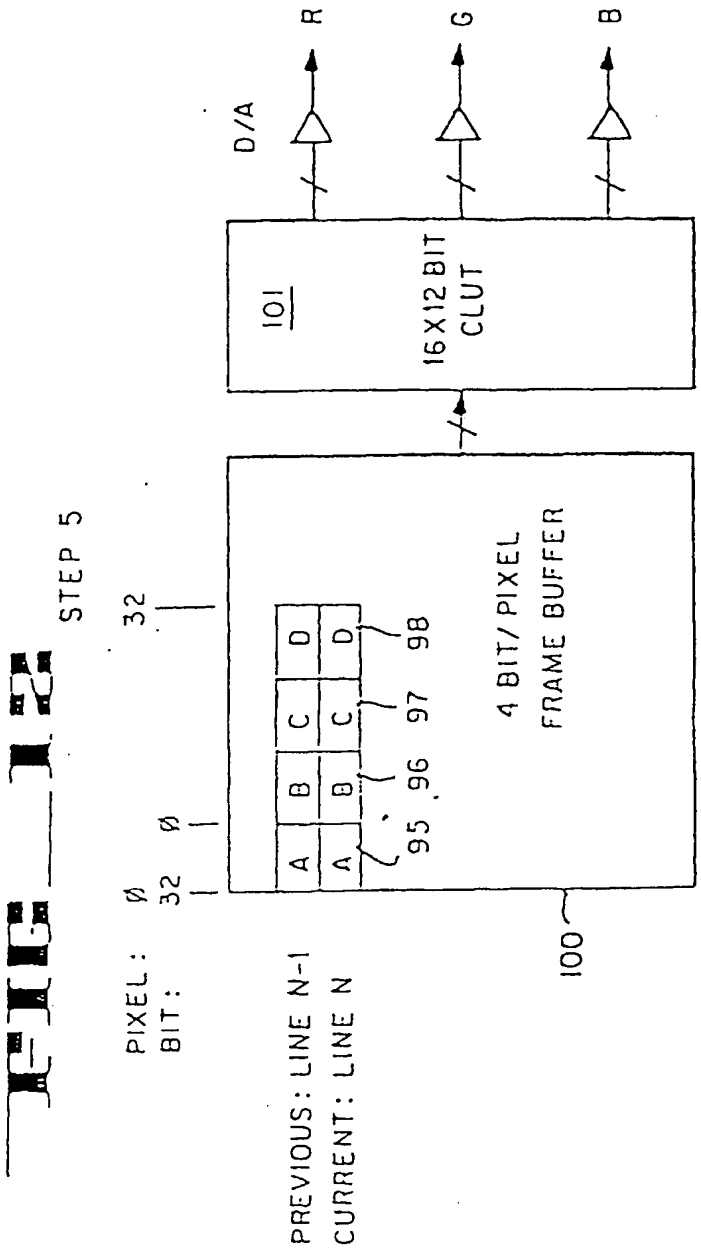

STEP 5
PIXEL: Ø
BIT: 32
Ø
32
A
B
C
D
95
96
97
98
PREVIOUS: LINE N-1
CURRENT: LINE N
4 BIT/ PIXEL
FRAME BUFFER
100
101
16X12 BIT
CLUT
D/A
R
G
B

FIG 13 GRAY VALUES LOADED INTO CLUT
(NON-GAMMA CORRECTED)

	WEIGHT:	25%	25%	50%	
	LINE:	N+1	N-1	N	
	PLANE:	2	1	Ø	CLUT GRAY VALUE
CLUT ADDRESS:		Ø	Ø	Ø	Ø% (BLACK)
		Ø	Ø	1	50%
		Ø	1	Ø	25%
		Ø	1	1	75%
		1	Ø	Ø	25%
		1	Ø	1	75%
		1	1	Ø	50%
		1	1	1	100% (WHITE)

1-2-1 CONVOLUTION KERNEL

WEIGHT		LINE
25%	1	N-1
50%	2	N
25%	1	N+1